CHAPITRE 5 : Données en table

CHAPITRE 5 : Données en table

1 Indexation des tables

1.1 Histoire

1964

- La première base de données est développée par l'armée américaine.
- C'est la première apparition du mot *database* qui signifie **base de données**.

Une base de données met en relation plusieurs tableaux à deux dimensions (un élément a une "abscisse" et une "ordonnée") appelées "tables".

1970

- Invention par le britannique **Edgard Codd** du **Système de Gestion de Bases de Données Relationnelles** (SGBDR).

1972

- Le **format csv** (***c****omma* ***s****eparated* ***v****alues*) permet d'enregistrer des tables de données dans un fichier texte et de l'échanger entre différents logiciels.

Sur une ligne les données sont séparées par un symbole séparateur.
Le symbole séparateur peut être une virgule (*comma* en anglais).

Exemple
Le fichier csv :

```
id,nom,race
1,Wanita,Tigre
2,Punk,Loutre
3,Toko,Toucan
4,Boubou,Elephant
5,Gustav,Ouistiti
```

représente la table de données :

id	nom	race
1	Wanita	Tigre
2	Punk	Loutre
3	Toko	Toucan
4	Boubou	Elephant
5	Gustav	Ouistiti

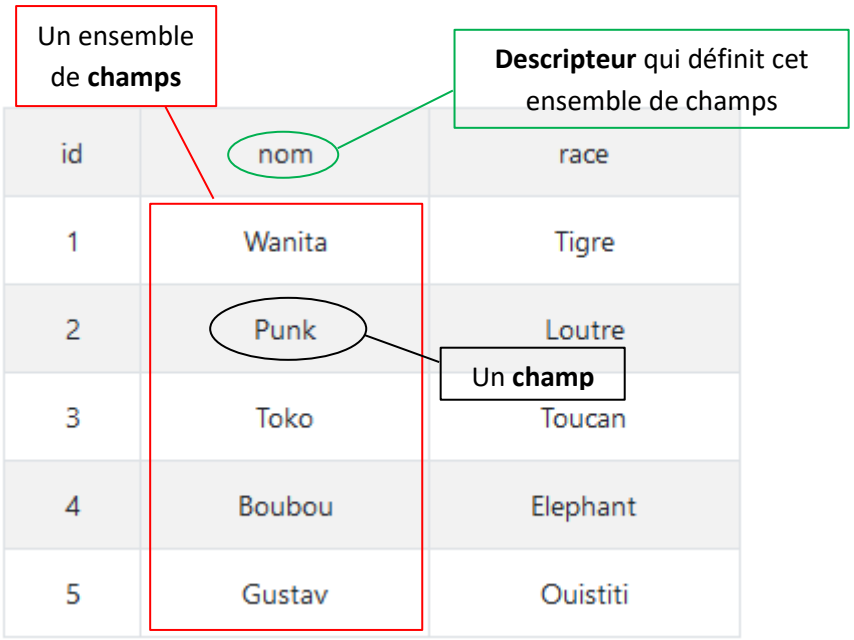
1979

- Commercialisation du **premier tableur** "Visicalc". Ont suivi les tableurs Excel ou Calc qui permettent de manipuler des données dans des tableaux.

Les tableurs peuvent importer des données depuis des fichiers au format csv.
Ensuite ils peuvent faire des traitements de ces données.
Enfin, ils peuvent exporter au format csv leurs résultats.

1.2 Vocabulaire

- Une **table** est un tableau à deux dimensions, représenté par une liste de listes en Python.

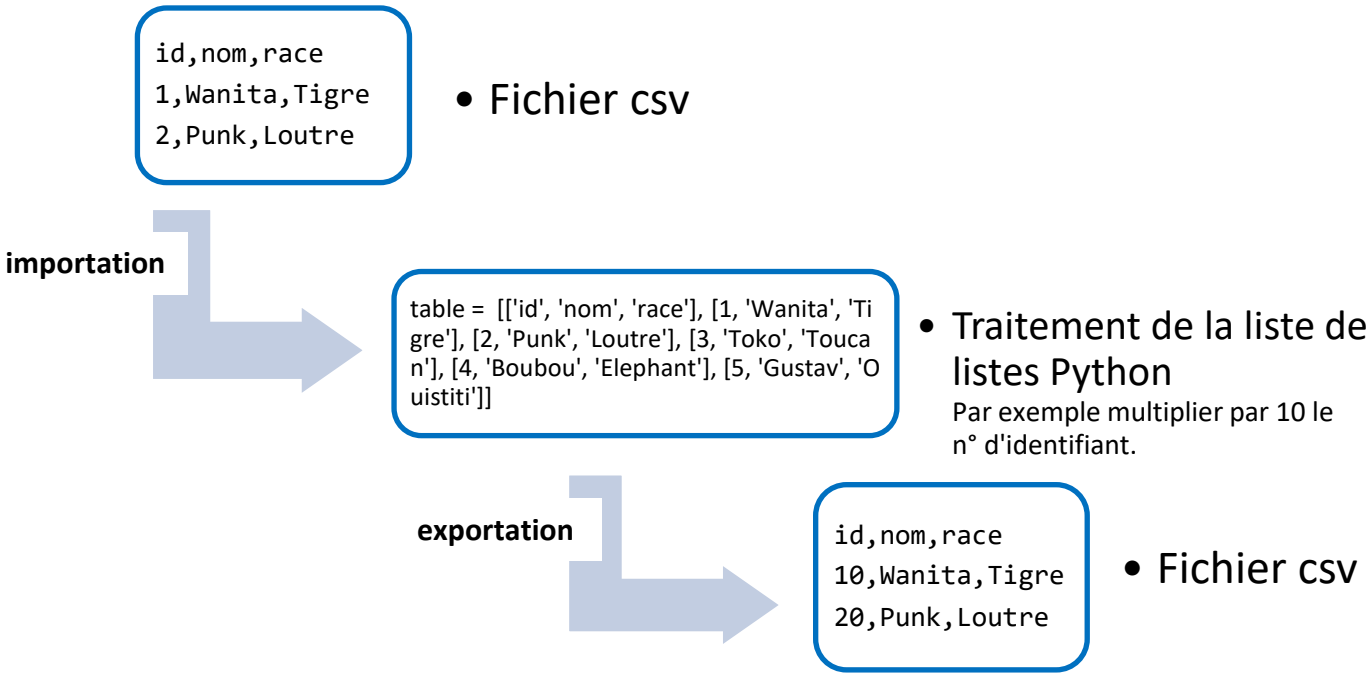


id	nom	race
1	Wanita	Tigre
2	Punk	Loutre
3	Toko	Toucan
4	Boubou	Elephant
5	Gustav	Ouistiti

Les descripteurs qui sont les noms des colonnes. Chaque colonne est un ensemble de champs remplis de données de types variés (chaînes de caractères, nombres entiers, nombres à virgule flottante...)

- Python **importe**[1] un fichier csv. Il convertit le format csv en liste de listes par exemple. Une fois le traitement terminé, Python fait une **exportation** du résultat sous la forme d'un fichier csv.

Exemple



[1] **Importation :** Voir les programmes d'**importation** et d'**exportation** en annexe à la fin de ce cours.

- Rappel sur les index de listes de listes

Si `table` est une liste de listes, par exemple :

```
table = [['id', 'nom', 'race'], [1, 'Wanita', 'Tigre'], [2, 'Punk', 'Loutre'],
 [3, 'Toko', 'Toucan'], [4, 'Boubou', 'Elephant'], [5, 'Gustav', 'Ouistiti']]
```

alors **chaque ligne** de `table` **est une des listes.** A chaque ligne correspond un index.

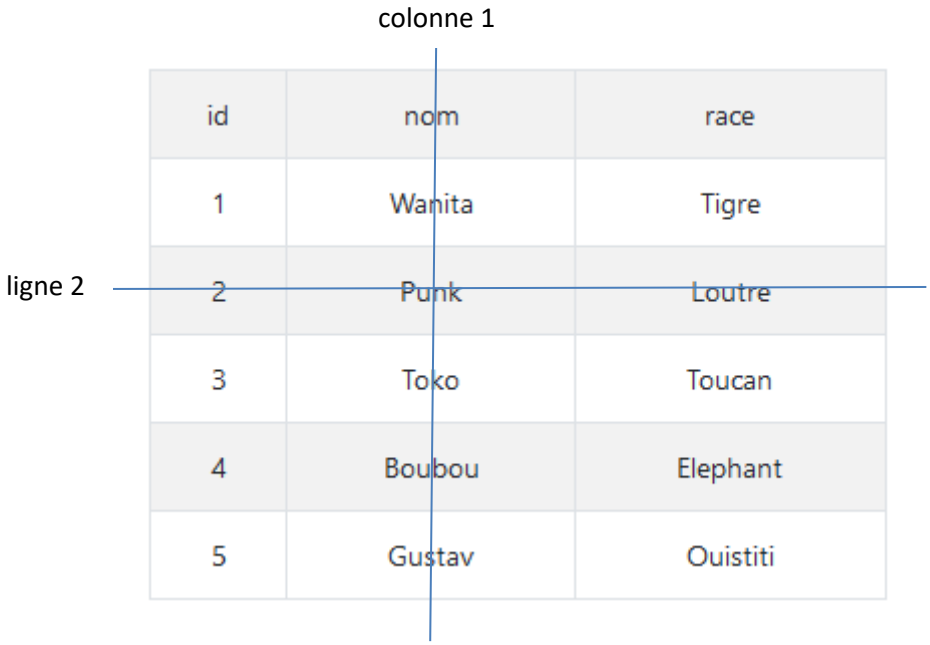
Exemple

```
table[0]    vaut   ['id', 'nom', 'race']
table[1]    vaut   ['1', 'Wanita', 'Tigre']
table[2]    vaut   ['2', 'Punk', 'Loutre']
```

Si on veut par exemple le nom de la loutre, il faut l'élément d'index 1 dans `table[2].` Donc il faut `table[2][1]` qui vaut `'Punk'.`

Ainsi `table[2][1]` est l'élément à l'intersection de la ligne 2 et de la colonne 1.

Les éléments sont doublement indexés, d'abord par le numéro de la ligne, ensuite par le numéro de l'élément dans cette ligne.



1.3 Autre représentation possible d'une table en Python

- Le **schéma** d'une table (sa structure) est donné par la première ligne, celle des descripteurs.

Exemple de schéma

`id` de type entier

`nom` de type chaîne de caractères

`race` de type chaîne de caractères

- Le **contenu** d'une table est donné par toutes les lignes suivantes, celles des champs.

Exemple de contenu

```
1,Wanita,Tigre

2,Punk,Loutre

3,Toko,Toucan

4,Boubou,Elephant

5,Gustav,Ouistiti
```

- Il est possible en Python de représenter la table de données d'une autre façon qu'une liste de listes. Il s'agit d'une **liste de dictionnaires**, avec **un dictionnaire pour chaque ligne** de la table.

Exemple

id	nom	race
1	Wanita	Tigre
2	Punk	Loutre
3	Toko	Toucan
4	Boubou	Elephant
5	Gustav	Ouistiti

Si cette table est représentée par une liste de dictionnaires alors elle vaut :

```
[{'id': 1, 'nom': 'Wanita', 'race': 'Tigre'}, {'id': 2, 'nom': 'Punk', 'race': 'Loutre'}, {
'id': 3, 'nom': 'Toko', 'race': 'Toucan'}, {'id': 4, 'nom': 'Boubou', 'race': 'Elephant'},
{'id': 5, 'nom': 'Gustav', 'race': 'Ouistiti'}]
```

Les clés des dictionnaires sont **les descripteurs**. Donc tous les dictionnaires ont les mêmes clés.

Les valeurs de chaque clé sont **les champs**.

2 Recherche de lignes dans une table

2.1 Sélection des lignes dans une table

Soit un **critère booléen** c'est à dire qui a pour valeur True ou False. Par exemple " La race est 'Tigre' ou la race est 'Toucan' ". On peut vouloir extraire toutes les lignes qui correspondent à ce critère.

Exemple 1 : Obtenir <u>la table</u> des animaux vérifiant un critère

On veut extraire la table pour laquelle le critère " La race est 'Tigre' ou la race est 'Toucan' " est vrai.

id	nom	race
1	Wanita	Tigre
2	Punk	Loutre
3	Toko	Toucan
4	Boubou	Elephant
5	Gustav	Ouistiti

- **Méthode 1 : Avec des fonctions**

```
def selectionner(table, critere):
    """
    Cette fonction renvoie une liste nommée selection qui contient
    les lignes pour lesquelles critere(table[i]) renvoie la valeur True.

    Paramètres :
    ------------
        table : De type liste de listes.
        critere : De type fonction.
                  Elle est appelée pour chaque ligne table[i] et renvoie True ou False.
    Retourne :
    ----------
        selection : De type liste.
                    C'est la liste des lignes qui correspondent au critère.
    """
    selection = []  # Initialisation la liste de retour par une liste vide.
    for i in range(1,len(table)): # Boucle pour toutes les lignes de la table sauf la 1ere.
        if critere(table[i]): # Si la fonction critere renvoie True pour la ligne table[i].
            selection.append(table[i]) # Ajoute la ligne i à la liste selection.
    return selection

table = [['id', 'nom', 'race'], [1, 'Wanita', 'Tigre'], [2, 'Punk', 'Loutre'], [3, 'Toko',
'Toucan'], [4, 'Boubou', 'Elephant'], [5, 'Gustav', 'Ouistiti']]

recherche = selectionner(table, lambda[2] x: x[2]=='Tigre' or x[2]=='Toucan')
```

Résultat

recherche vaut [[1, 'Wanita', 'Tigre'], [3, 'Toko', 'Toucan']]

---

[2] **lambda :** Une fonction lambda est une fonction d'une seule ligne déclarée sans nom, qui peut avoir un nombre quelconque d'arguments. Il arrive souvent qu'une fonction lambda soit passée en argument à une autre fonction. Dans cette fonction lambda, **x est du type liste** et a pour valeur *n'importe quelle ligne* de la table.

- **Méthode 2 : Avec une liste en compréhension et filtrage booléen**

[*expression* for x in *valeurs de départ* if *booléen de filtrage*]

```
recherche = [ligne for ligne in table if ligne[2]=='Tigre' or ligne[2]=='Toucan']
```

Résultat
recherche vaut `[[1, 'Wanita', 'Tigre'], [3, 'Toko', 'Toucan']]`

Exemple 2 : Obtenir la liste des noms des animaux vérifiant un critère

- **Méthode 1 : Avec des fonctions**

La liste des descripteurs de la table est `['id', 'nom', 'race']`.

Ainsi le nom est le champ d'index 1 dans chaque ligne de la table.

Il faut donc **modifier** la cinquième ligne de **la fonction `selectionner(table, critere)`** ainsi :

Après modification :

```
def selectionner(table, critere):
    selection = []  # Initialisation la liste de retour par une liste vide.
    for i in range(1,len(table)): # Boucle pour toutes les lignes de la table sauf la 1ere.
        if critere(table[i]): # Si la fonction critere renvoie True pour la ligne table[i].
            selection.append(table[i][1]) # Ajoute le champ 1 de la ligne i à selection.
    return selection

recherche_noms = selectionner(table, lambda x: x[2]=='Tigre' or x[2]=='Toucan')
```

Résultat
recherche_noms vaut `['Wanita', 'Toko']`

- **Méthode 2 : Avec une liste en compréhension et filtrage booléen**

```
recherche = [ligne[1] for ligne in table if ligne[2]=='Tigre' or ligne[2]=='Toucan']
```

Résultat
recherche vaut `['Wanita', 'Toko']`

2.2 Tests de cohérence

- Tester **la cohérence** consiste à lire toute la table et vérifier si les données sont du type qui correspond au descripteur.

Exemples

Tous les champs du descripteur 'id' sont-ils des entiers ?
Tous les champs du descripteur 'nom' sont-ils des chaînes de caractères ?
Tous les champs du descripteur 'race' sont-ils des chaînes de caractères ?

2.3 Recherche de doublons

- Rechercher les **doublons**, c'est rechercher s'il y a deux lignes rigoureusement identiques, hormis leur identifiant.

Exemple

```
table = [['id', 'nom', 'race'], [1, 'Wanita', 'Tigre'], [2, 'Punk', 'Loutre'],\
         [3, 'Toko', 'Toucan'], [4, 'Boubou', 'Elephant'], [5, 'Gustav', 'Ouistiti'],\
         [6, 'Ali', 'Toucan'], [7, 'Boubou', 'Elephant'], [8, 'Bubu', 'Tortue']]
```

est la table des animaux. Elle contient un doublon. Dans ce cas on supprimera une des deux lignes de manière à ne laisser qu'un exemplaire de la ligne. Mais il faut d'abord détecter les doublons.

```
def detecter_doublons(table):
    """
    Cette fonction renvoie une liste nommée doublons qui contient les numéros
    des lignes sous forme [i, j] telles que table[i] et table[j] sont identiques,
    en dehors de leur 'id'.

    Paramètres :
    ------------
        table : De type liste de listes.
    Retourne :
    ----------
        doublons : De type liste.
                   C'est la liste des lists [i, j] des doublons.
    """

    doublons = []
    for i in range (1, len(table)):
        for j in range(i + 1, len(table)):
            if table[i][1:] == table[j][1:]:  #  Si tous les champs sont égaux
                                              #   depuis l'index 1 jusqu'à la fin.
                doublons.append([i, j])
    return doublons
```

Résultat

`detecter_doublons(table)` renvoie `[[4, 7]]`

3 Tri d'une table selon l'ordre d'une colonne

- Tout d'abord, le **tri d'une liste simple** en Python peut se faire :

- Avec un algorithme de tri écrit à la main.

- En utilisant la fonction `sorted()`. Cette fonction native en Python (c'est à dire qu'on n'a pas besoin d'importer de bibliothèque) crée une copie de la liste, et trie cette copie selon l'ordre croissant.

Exemple

Après exécution de :	on obtient :
`ma_liste = [2, 35, 42, 39, 41]` `ma_liste_croissante = sorted(ma_liste)`	`ma_liste` `[2, 35, 42, 39, 41]` `ma_liste_croissante` `[2, 35, 39, 41, 42]`

- Ensuite, le **tri d'une liste de listes** en Python peut se faire à condition de préciser selon quelle colonne (descripteur) on veut trier l'ordre des lignes

Exemple

Après exécution de :

```
animaux =  [['id', 'nom', 'race'], [1, 'Wanita', 'Tigre'],\
            [2, 'Punk', 'Loutre'], [3, 'Toko', 'Toucan'], \
            [4, 'Boubou', 'Elephant'], [5, 'Gustav', 'Ouistiti']]

animaux_selon_nom = sorted(animaux, key=lambda x: x[1])
```

On obtient :

```
animaux_selon_nom
[[4, 'Boubou', 'Elephant'], [5, 'Gustav', 'Ouistiti'],\
 [2, 'Punk', 'Loutre'], [3, 'Toko', 'Toucan'],\
 [1, 'Wanita', 'Tigre'], ['id', 'nom', 'race']]
```

Remarque

Le deuxième argument de la fonction `sorted()` se nomme `key`. C'est une fonction lambda[3] qui à x (dans cet exemple **x est une liste** représentant une ligne quelconque) fait correspondre l'élément d'index 1 dans cette ligne.

Problème : Les lignes ont été triées selon **l'ordre lexicographique**[4] des noms, mais cela a placé la ligne des descripteurs à la fin. En effet le 'n' de **nom** est classé après les majuscules des noms.

---

[3] **lambda :** Une fonction lambda est à nouveau utilisée comme argument *key* dans la fonction `sorted()`. Le rôle de *key* est de **préciser** selon l'ordre de **quelle colonne** la fonction `sorted()` doit ordonner les lignes.

[4] **Ordre lexicographique :** Ordre des caractères suivant leur point de code Unicode. L'espace puis les nombres en premier, puis les majuscules non accentuées, puis les minuscules non accentuées, puis les lettres accentuées.

4	Boubou	Elephant
5	Gustav	Ouistiti
2	Punk	Loutre
3	Toko	Toucan
1	Wanita	Tigre
id	nom	race

- Pour éviter ce problème, il suffit de demander de trier sur `animaux[1:]`.

`animaux[1:]` signifie qu'on fait un *slicing* depuis l'élément d'index 1 jusqu'à la fin de la liste (puisqu'on n'a pas précisé l'index de fin).

Autrement dit on retire l'élément d'index 0 c'est à dire la ligne des descripteurs.

Exemple

Après exécution de :

```
animaux =  [['id', 'nom', 'race'], [1, 'Wanita', 'Tigre'],\
            [2, 'Punk', 'Loutre'], [3, 'Toko', 'Toucan'], \
            [4, 'Boubou', 'Elephant'], [5, 'Gustav', 'Ouistiti']]

animaux_selon_nom = sorted(animaux[1:], key=lambda x: x[1])
```

On obtient :

```
animaux_selon_nom
[[4, 'Boubou', 'Elephant'], [5, 'Gustav', 'Ouistiti'],\
 [2, 'Punk', 'Loutre'], [3, 'Toko', 'Toucan'],  [1, 'Wanita', 'Tigre']
```

4	Boubou	Elephant
5	Gustav	Ouistiti
2	Punk	Loutre
3	Toko	Toucan
1	Wanita	Tigre

4 Fusion de deux tables

Définition

La fusion consiste à combiner deux tables `t1` et `t2` pour n'en former plus qu'une seule.

Si les deux tables **ont les mêmes descripteurs** (dans le même ordre) alors il est possible de les **concaténer**, c'est à dire d'ajouter les lignes de `t2` à celle de `t1` pour former `t1 + t2`.

Exemple :

Après exécution de :

```
t1 =  [['id', 'nom', 'race'],\
           [1, 'Wanita', 'Tigre'], [2, 'Punk', 'Loutre']]

t2 =  [['id', 'nom', 'race'],\
           [3, 'Toko', 'Toucan'], [4, 'Boubou', 'Elephant']]

t3 = t1 + t2  #  Concaténation des deux tables.
```

On obtient :

```
t3
[['id', 'nom', 'race'], [1, 'Wanita', 'Tigre'], [2, 'Punk', 'Loutre'],\
 ['id', 'nom', 'race'], [3, 'Toko', 'Toucan'], [4, 'Boubou', 'Elephant']]
```

id	nom	race
1	Wanita	Tigre
2	Punk	Loutre
id	nom	race
3	Toko	Toucan
4	Boubou	Elephant

Problème : Les lignes des tables ont bien été concaténées, mais la ligne des descripteurs en provenance de `t2` est en trop.

- Pour éviter ce problème, il suffit de concaténer en enlevant à `t2` sa première ligne (celle des descripteurs). On utilise le *slicing* comme précédemment.

Après exécution de :

```
t3 = t1 + t2[1:]  #  Concaténation des deux tables en enlevant à t2 sa première ligne.
```

On obtient :

```
[['id', 'nom', 'race'], [1, 'Wanita', 'Tigre'], [2, 'Punk', 'Loutre'],\
 [3, 'Toko', 'Toucan'], [4, 'Boubou', 'Elephant']]
```

Avant de concaténer deux tables ayant les mêmes descripteurs, il faut s'assurer que les domaines de valeurs des champs sont les mêmes. Par exemple, s'il s'agit de grandeurs physiques, alors elles doivent être dans les mêmes unités. Ou s'il s'agit d'effectifs, ils doivent être exprimés de la même façon.

Exemple

A partir de `t1`

nom	continent	superficie	population
Afghanistan	Asie	652664	34125
Angola	Afrique	1264700	30356

et de `t2`

nom	continent	superficie	population
Albabnie	Europe	28750	3048000
Andorre	Europe	468	85600

on a obtenu

`t3 = t1 + t2[1:]`

nom	continent	superficie	population
Afghanistan	Asie	652664	34125
Angola	Afrique	1264700	30356
Albabnie	Europe	28750	3048000
Andorre	Europe	468	85600

- Les descripteurs sont bien les mêmes, dans le même ordre. Mais le problème vient du fait que dans la table de données `t2` (pays européens) la population est exprimée en nombre d'habitants, tandis qu'elle est exprimée *en milliers d'habitants* pour les autres pays.

Avant de concaténer, il aurait fallu diviser par 1000 les champs de la colonne population dans `t2`.

5 Annexe

5.1 Programme d'importation d'un fichier csv

```
import csv

def lecture_fichier(nom_fichier):
    """
    Importe un fichier csv existant et le transforme en liste de listes.

    Paramètres :
    ------------
        nom_fichier : Du type chaîne de caractères.
                      C'est le nom du fichier csv (encodé en utf-8),
                      par exemple 'fruits.csv'
    Renvoie :
    ---------
              Une liste de listes.
    """

    with open(nom_fichier, mode='r', encoding='utf-8-sig') as fichier_ouvert:
        return [ligne for ligne in csv.reader(fichier_ouvert)]
```

Exemple

On saisit dans un tableur le tableau :

	A	B	C	D	E	F
1	id	Rouge	Orange	Jaune	Rond	Allonge
2	Tomate	1	0	0	1	0
3	Banane	0	0	1	0	1
4	Citron	0	0	1	1	0
5	Pomme	1	0	0	1	0
6	Peche	0	1	0	1	0

On l'exporte au format csv en UTF-8

On obtient le fichier fruit.csv qui peut être ouvert dans le logiciel éditeur de textes bloc-notes :

id;Rouge;Orange;Jaune;Rond;Allonge

Tomate;1;0;0;1;0

Banane;0;0;1;0;1

Citron;0;0;1;1;0

Pomme;1;0;0;1;0

Peche;0;1;0;1;0

Dans bloc-notes on remplace les points-virgules par des virgules, en allant dans le menu Edition puis Remplacer puis Remplacer tout :

id,Rouge,Orange,Jaune,Rond,Allonge

Tomate,1,0,0,1,0

Banane,0,0,1,0,1

Citron,0,0,1,1,0

Pomme,1,0,0,1,0

Peche,0,1,0,1,0

On ré enregistre au fruits.csv **dans le même dossier que celui où se trouve le programme Python précédent.**

On exécute le programme Python.

On obtient

```
mon_tableau = lecture_fichier('fruits.csv')
print(mon_tableau)
```

```
[['id', 'Rouge', 'Orange', 'Jaune', 'Rond', 'Allonge'], ['Tomate', '1', '0',
'0', '1', '0'], ['Banane', '0', '0', '1', '0', '1'], ['Citron', '0', '0', '1',
'1', '0'], ['Pomme', '1', '0', '0', '1', '0'], ['Peche', '0', '1', '0', '1',
'0']]
```

Ainsi le tableau

	A	B	C	D	E	F
1	id	Rouge	Orange	Jaune	Rond	Allonge
2	Tomate	1	0	0	1	0
3	Banane	0	0	1	0	1
4	Citron	0	0	1	1	0
5	Pomme	1	0	0	1	0
6	Peche	0	1	0	1	0

a été transformé en fichier csv puis importé en liste de listes.

5.2 Programme d'exportation d'un fichier csv

Exemple

```
import csv

def exporter(tableau, nom_fichier):
    """
    Cette fonction exporte une liste de listes sous la forme csv.

    Paramètres :
    ------------
        tableau : Du type liste de listes.
        nom_fichier : Du type chaîne de caractères.
                      C'est le nom donné a fichier de sortie,
                      par exemple 'capitales1.csv'

    """

    with open(nom_fichier, mode='w', newline='') as fichier_ouvert:
        csv.writer(fichier_ouvert).writerows(tableau)


mon_tableau = [['id', 'Rouge', 'Orange', 'Jaune', 'Rond', 'Allonge'],\
               ['Tomate', '1', '0', '0', '1', '0'],\
               ['Banane', '0', '0', '1', '0', '1'],\
               ['Citron', '0', '0', '1', '1', '0'],\
               ['Pomme', '1', '0', '0', '1', '0'],\
               ['Peche', '0', '1', '0', '1', '0']]



exporter(mon_tableau, 'fichier_exporte.csv')
```

Résultat

Le fichier `fichier_exporte.csv` est apparu dans le même répertoire que celui du fichier Python. Si on ouvre fichier_exporte.csv avec l'éditeur de textes bloc- notes, on a :

id,Rouge,Orange,Jaune,Rond,Allonge
Tomate,1,0,0,1,0
Banane,0,0,1,0,1
Citron,0,0,1,1,0
Pomme,1,0,0,1,0
Peche,0,1,0,1,0

Dans le tableur Excel, dans le menu Données se trouve la commande "A partir d'un fichier Texte / CSV". On peut alors charge le fichier CSV sous forme d'un tableau.